Detecting Runtime Business Process Compliance with Artifact Lifecycles

Qi He[1,2]

[1] School of Computer Science, Fudan University, China
[2] College of Information Technology, Shanghai Ocean University, China
061021059@fudan.edu.cn

Abstract. Detecting business process compliance in runtime is the complementary to static compliance checking in the stage of process design, and allows checking whether an execution of a business process satisfies a given constraint. In this paper, runtime compliance checking is used for artifact-centric business process and artifact lifecycles are treated as business constraints. Previous methods for runtime compliance checking mainly put focus on activities in business process and lose the attention for data. In this work we concentrate on both the evolution of artifacts (data) and services (activities) to identify the frontier between decidability and undecidability of the runtime compliance problem. We also provide decidable results and the implement method under regular and context-free artifact lifecycles.

Keywords: BPM, artifact, compliance, decidability.

1 Introduction

As the technique to ensure business processes conforming to rules and regulations, compliance checking occurs in almost full stages of BPM (business processes management) lifecycle. More and more enterprises and organizations are focusing on employing this technique into their BPM systems to improve business efficiency.

Traditionally, the BPM systems are organized in control-centric business process models in which activities are focused on and data just serve as inputs and outputs of some services. In recent years, as a data-centric approach, the artifact-centric methodology[1] has emerged as a new paradigm to support business process management. In the future, most business processes will be implemented based on the artifact-centric idea and this methodology will become the major tendency of BPM.

Currently, researches on compliance of artifact-centric business processes have been carried out by guaranteeing business process models complying with business constraints in the process design phase [2,3,4]. But, it is not comprehensive to only consider compliance during the process of design. For example, in many business processes, non-compliant behaviors can most likely emerge during business process executions for human errors. So, how to detect runtime business process compliance is a crucial challenge of business process management.

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 426–432, 2013.

As key information records in business process, artifacts include both the business-relevant data and their own lifecycles which constrain how they can evolve over time from being created to being achieved as the result of services being applied to them. In general, artifact lifecycles reflect rules that constrain business process. Therefore artifact lifecycles can be regarded as business constraints.

In this paper, our goal is thus to discuss the problem of detecting runtime compliance between artifact-centric business processes executions and business constraints (artifact lifecycles). We propose the formal definition of artifact lifecycles which are languages over special alphabets. Based on this definition, we define the problem of runtime compliance as the acceptance problem for languages. Then we identify the frontier between decidability and undecidability of this problem. To obtain various decidability results, we focus on two kinds of business constraints, regular and context-free artifact lifecycles, and provide the result of decidability. To our knowledge, the present work is the first to study the runtime compliance problem by considering both data (artifacts) and activities (services) formally.

The paper is organized as follows. In Section 2 we survey related works. In Section 3 we formulate the problem of runtime compliance checking with artifact lifecycles. In Section 4 we study the decidability of runtime compliance checking, provide decidable results and the implement method under regular and context-free artifact lifecycles. Section 5 concludes the paper.

2 Related Work

Compliance checking for artifact-centric business processes can be executed both in the stage of process design and the stage of process implementation.

In the stages of process design, compliance checking is developed to verify if business process models is complied with business constraints. [2,3] puts attentions on static analyzing the properties of artifact-centric business processes, such as general temporal constraints. [4] discusses static checking problem, based on the conditions that business processes are represented in artifact systems and the rules are expressed in linear-time temporal logic.

In the stages of process implementation, the runtime compliance checking usually utilizes the results of business process executions to judge if the operations do not violate business rules. In [5, 6], the problem of conformance checking is discussed, which is to check whether the business process executions recorded in logs is consistent with the business process models. [7] presents a novel runtime verification framework based on linear temporal logic, and translates constraint model into colored automata to monitor the business process execution. But, these techniques for runtime compliance checking only consider the activities of business and omit the evolution of data entities.

In this paper, we propose a novel approach to implement runtime compliance checking. And in our works, the changes of data and activities are all considered.

3 Problem Statement

For formulating the problem of runtime compliance for artifact-centric business processes, we provide the definitions of artifact schema, artifact instance, and artifact. We assume the existence of the infinite set $D = \{D_1, D_2, \ldots, D_i, \ldots\}$, where D_i is a finite domain.

Definition 1. An *artifact schema* Γ (simply schema) is a tuple of (U, τ), where (1)U is a finite set of attributes, a special attribute $ID \in U$ is identifier attribute, and (2)$\tau: U \rightarrow D$ is a total mapping.

Definition 2. An *artifact instance* a_Γ of schema Γ is a tuple of (id, μ), where (1) μ is a partial mapping that assigns each attribute X in U a value x, $x \in \tau(X)$, and (2) $id \in \tau(ID)$ is an identifier.

Definition 3 [8]**.** An *artifact* A_Γ of schema Γ is a tuple of (id, T, AI, λ), where (1) id is the identifier of artifact, (2) $T=\{t_1, t_2, \ldots, t_n\}$ is the domain of time, (3) AI is the set of artifact instances of Γ, the identifiers of which are id, (4) λ: $T \rightarrow AI$ is a total mapping.

Example 1. We briefly describe an example of a business process for equipment sale in shops to illustrate concepts as we introduce them. In this application, only one artifact schema *Order*, shown in Table 1, is included. Table 2 and Table 3 show some artifact instances and artifacts of *Order*.

Table 1. Artifact Schema *Order*

U	$\tau(X), X \in U$
ID	{ 01, 02 }
equipName	{printer, displayer }
Customer	{ Tom, Jack}
checkAvail	{yes, no}
checkPaied	{yes, no}

Table 2. Artifact Instances of *Order*

Artifact Instances Name	*id*	*equipName*	*Customer*	*checkAvail*	*checkPaid*
a_{11}	01	printer	Tom	null	null
a_{12}	01	printer	Tom	yes	null
a_{13}	01	printer	Tom	yes	yes
a_{21}	02	displayer	Jack	null	null
a_{22}	02	displayer	Jack	no	null
a_{23}	02	displayer	Jack	yes	null
a_{24}	02	displayer	Jack	yes	yes

Table 3. Artifacts of Order

Artifact Name	*id*	*equipName*	*Customer*	*checkAvail*	*checkPaid*	*TimeStamp*
A_1	01	printer	Tom	null	null	20-10,09:12
	01	printer	Tom	yes	null	21-10,10:11
	01	printer	Tom	yes	yes	21-10,15:09
A_2	02	displayer	Jack	null	null	20-10,19:05
	02	displayer	Jack	no	null	21-10,10:30
	02	displayer	Jack	yes	null	23-10,10:05
	02	displayer	Jack	yes	yes	23-10,13:15

Services are used to modify the values of artifact attributes. The evolutions of artifacts are recorded in execution logs in form of sequences made up of alternate services and artifact instances. In this paper, we define the sequence as a string over an alphabet.

Definition 4. A *service artifact-instance string* ω (simply s-a string) of Γ is a string over the alphabet Σ, where $\Sigma \subseteq S \times AI$, AI is a set of artifact instances of Γ, S is a set of services acting on artifacts of Γ and artifact instances occurring in the string have the same identifier.

Example 2. For the artifact schema *Order* in Example 1, there exist the set of services S_{Order} and the set of artifact instances AI_{Order}, where $S_{Order} = \{s_{o1}, s_{o2}, s_{o3}\}$,

s_{o1}: Create artifacts of *Order*; s_{o2}: Check whether equipments are available,

s_{o3}: Check whether the order is paid,

and $AI_{Order} = \{a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{24}\}$. Let alphabet $\Sigma_{Order} \subseteq S_{Order} \times AI_{Order}$, and $\Sigma_{Order} = \{(s_{o1}, a_{11}), (s_{o1}, a_{21}), (s_{o2}, a_{12}), (s_{o2}, a_{22}), (s_{o2}, a_{23}), (s_{o3}, a_{13}), (s_{o3}, a_{24})\}$. ω_1, ω_2, ω_3 are three s-a strings of *Order*, where

$\omega_1 = (s_{o1}, a_{11})(s_{o2}, a_{12})(s_{o3}, a_{13})$, $\omega_2 = (s_{o1}, a_{21})(s_{o2}, a_{22})(s_{o2}, a_{23})(s_{o3}, a_{24})$,

$\omega_3 = (s_{o1}, a_{21})(s_{o2}, a_{22})(s_{o2}, a_{22})(s_{o2}, a_{23})(s_{o3}, a_{24})$.

Next, we introduce two operations which are provided to get artifact instance sequences (simply AISs) from an artifact and from an s-a string respectively.

Definition 5. Given an artifact $A_\Gamma = (id, T, AI, \lambda)$ of schema Γ and a set of AISs Q in which a AIS is sequence of artifact instances in AI, we provide *the operation* α, $\alpha: \{A_\Gamma\} \rightarrow Q$, where $\alpha(A_\Gamma)$ is a AIS $a_1 a_2 \ldots a_i \ldots a_n$ and $a_i = \lambda(t_i)$, $0 \leq i \leq n$.

Definition 6. Let $\omega = (s_1, a_1)(s_2, a_2)\ldots(s_i, a_i)\ldots(s_n, a_n)$ be a s-a string over $\Sigma \subseteq S \times AI$ and Q be a set of AISs in which a AIS is sequence of artifact instances in AI. We provide *the operation* β, $\beta: \{\omega\} \rightarrow Q$, where $\beta(\omega)$ is the AIS $a_1 a_2 \ldots a_i \ldots a_n$.

Example 3. We apply the operation α to A_1, A_2 in Example 1 and apply β to ω_1, ω_2, ω_3 in Example 3. The results are following.

$\alpha(A_1) = a_{11} a_{12} a_{13}$, $\alpha(A_2) = a_{21} a_{22} a_{23} a_{24}$,

$\beta(\omega_1) = a_{11} a_{12} a_{13}$, $\beta(\omega_2) = a_{21} a_{22} a_{23} a_{24}$, $\beta(\omega_3) = a_{21} a_{22} a_{22} a_{23} a_{24}$

Definition 7. Given an artifact A_Γ of schema Γ and a s-a string ω of Γ, ω is the *evolution* of A_Γ if $\alpha(A_\Gamma) = \beta(\omega)$.

Now, we can say that $\omega_1$is the evolution of artifact A_1, and ω_2 is the evolution of artifact A_2.

Definition 8. Let AI be a set of artifact instances of schema Γ and S is a set of services acting on artifacts of Γ. An *artifact lifecycle* L of Γ is a language over alphabet Σ, $\Sigma \subseteq S \times AI$, i.e., $L = \{\omega \mid \omega \text{ is a s-a string of } \Gamma\}$.

Example 4. Suppose that $L_{Order} = \{\omega_1, \omega_2, \omega_3\}$, and ω_1, ω_2, ω_3 are s-a strings of Γ shown in Example 3. Then L_{Order} is an artifact lifecycle of Γ.

According to the above definitions, we can know that the execution of an artifact-centric business process can be recorded in form of the evolutions of artifacts, and that the problem of runtime compliance is equivalent to the membership problem for an artifact lifecycle L. Formally, we state the problem as follows:

Runtime Compliance Problem: Suppose that Γ is an artifact schema, A_Γ is an artifact of Γ. Given an artifact lifecycle L of Γ and the evolution ω of A_Γ, we say that the execution of business process on artifact A_Γ is in accordance with artifact lifecycle, if $\omega \in L$ is hold.

4 Detecting Runtime Business Process Compliance

Firstly, we address the decision problem of Runtime Compliance Problem. As illustrated in Section 3, an artifact lifecycle is an arbitrary language over alphabet $\Sigma \subseteq S \times AI$, then we show that some artifact lifecycles are not Turing-recognizable.

Theorem 1. Let Γ be an artifact schema, AI be a set of artifact instances of schema Γ and S be a set of services acting on artifacts of Γ. Some artifact lifecycles of Γ are not Turing-recognizable.

Clearly, if an artifact lifecycle is an arbitrary language over alphabet Σ, Runtime Compliance Problem is probable not Turing-recognizable for the reason that the artifact lifecycle is not Turing-recognizable. Therefore we consider the case that the artifact lifecycle is described in a Turing machine.The artifact lifecycle described in a Turing machine is called TM artifact lifecycle in this paper.

Theorem 2. Let Γ be an artifact schema, A_Γ be an artifact of Γ, L_M be a TM artifact lifecycle of Γ and L_M be described in a Turing machine M. Given L_M and the evolution ω of A_Γ, Runtime Compliance Problem is undecidable.

Proof Idea. This proof is obvious, because L_M described in a Turing machine M and the acceptance problem for Turing machine is undecidable. □

Theorem 1 shows that there are artifacts whose lifecycles are not Turing-recognizable, and Theorem 2 implies that Runtime Compliance Problem is undecidable under the condition that an artifact lifecycle is described in Turing machine. To obtain various decidability results, we focus on artifact lifecycles that are described in regular expressions and pushdown automatons.

Definition 9. Let L_1 and L_2 be artifact lifecycles. We define the regular operations *union*, *concatenation*, and *star* as follows:

1. Union. $L_1 \cup L_2 = \{x \mid x \in L_1 \text{or } x \in L_2,\}$.
2. Concatenation. $L_1 \circ L_2 = \{xy \mid x \in L_1, y \in L_2,$ and the artifact instances occurring in x and y have the same identifier$\}$.
3. Star. $L_1^* = \{x_1 x_2 \ldots x_k \mid k \geq 0,$ each $x_i \in L_1,$ and the artifact instances occurring in $x_1, x_2, \ldots, x_k$ have the same identifier$\}$.

Definition 10. Let Γ be an artifact schema, AI be a set of artifact instances of schema Γ and S is a set of services acting on artifacts of Γ. For an alphabet $\Sigma \subseteq S \times AI$, say that

R is a *regular artifact lifecycle expression* of Γ (simply regular ALE) over Σ if R is any form listed below.

1. (s, a) for some (s, a) in the alphabet Σ,
2. ε,
3. $\varnothing$,
4. $(R_1 \cup R_2)$, where $R_1$and R_2 are regular ALEs,
5. $(R_1 \circ R_2)$, where $R_1$and R_2 are regular ALEs,
6. (R_1^*), where R_1 is a regular ALE.

An artifact lifecycle described in a regular ALE is a regular artifact lifecycle.

Theorem 3. Let Γ be an artifact schema, A_Γ be an artifact of Γ, L_R be a regular artifact lifecycle of Γ and be described in a regular ALE R. Given L_R and the evolution ω of A_Γ, Runtime Compliance Problem is decidable.

The regular ALE provides us a powerful tool to describe artifact lifecycles. But the fact still exists that some artifact lifecycles cannot be described in this way. So, we present pushdown automata (PDA) as more powerful tools to describe artifact lifecycles, and call artifact lifecycles described in pushdown automata context-free artifact lifecycles.

Definition 11. Let Γ be an artifact schema, AI be a set of artifact instances of schema Γ and S be a set of services acting on artifacts of Γ. A *pushdown automata N* of Γ is a 6-tuple$(Q, \Sigma, Z, \delta, q_0, F)$, where

1. Q is the set of finite states,
2. Σ is the finite input alphabet, $\Sigma \subseteq S \times AI$,
3. Z is the finite stack alphabet,
4. δ: $Q \times \Sigma_\varepsilon \times Z_\varepsilon \to P(Q \times Z_\varepsilon)$, where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $Z_\varepsilon = Z \cup \{\varepsilon\}$, and $P(Q \times Z_\varepsilon)$ is the power set of $Q \times Z_\varepsilon$,
5. $q_0 \in$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The language recognized by PDA N is a context-free artifact lifecycles L_N of Γ.

Theorem 4. Let Γ be an artifact schema, A_Γ be an artifact of Γ, L_N be a context-free artifact lifecycle of Γ and be described in a PDA N. Given L_N and the evolution ω of A_Γ, Runtime Compliance Problem is decidable.

We applied our approach to detecting business process compliance with artifact lifecycles without considering state explosion problems. Here, we provide the framework of this method below:

1. Describe business constraints in regular artifact lifecycles or context-free artifact lifecycles;
2. Generate the corresponding language accepters from the business constraints;
3. Abstract the service artifact-instance string ω (s-a string) from business process execution logs, and ensure ω is the evolution of A_Γ;

4. Provide ω to accepters. If ω is accepted, we can return the result that the execution of business process on artifact A_Γ is in accordance with artifact lifecycles.

5 Summary and Future Work

This paper studies the runtime compliance checking for artifact-centric business process. Based on the approach in this paper, we can effectively check whether business processes are compliant with artifact lifecycles in runtime. In future, we are going to build up a comprehensive compliance checking environment for BPM.

References

1. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
2. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
3. Gerede, C.E., Su, J.: Specification and Verification of Artifact Behaviors in Business Process Models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
4. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: International Conference on Database Theory (ICDT 2009), pp. 252–267. ACM Press (2009)
5. Rozinat, A., Jong, I., Gunther, C., Aalst, W.: Conformance Analysis of ASML's Test Process. In: GRCIS 2009, vol. 459, pp. 1–15. CEUR-WS.org (2009)
6. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011)
7. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011)
8. Ying, W., Guohua, L., Zhen, H., et al.: The Research on Validity of Artifact in BPM. In: International Conference on Business Management and Electronic Information, pp. 15–18. IEEE, Piscataway (2011)